从Linux文件系统到Hadoo分布式文件系统hdfs

2017-04-18 | 191

大数据开发的同学的日常工作是绕不开Hadoop生态圈的，每天打交道最多的就是hadoop上的分布式文件系统hdfs了。那么，为什么hdfs会称为一种“文件系统”呢？仅仅是因为他的命令与服务器上的Linux操作系统的文件系统中各类命令类似或相同吗？那么hdfs又与Linux上的文件系统FS有什么异同呢？下面我们就来聊一聊这些问题。

Linux文件系统

研究Linux的文件系统，我们首先要引出Linux Fs下的一个重要概念——inode

什么是inode

理解inode，要从文件储存说起。
文件储存在硬盘上，硬盘的最小存储单位叫做”扇区”（Sector）。每个扇区储存512字节（相当于0.5KB）。
操作系统读取硬盘的时候，不会一个个扇区地读取，这样效率太低，而是会一次性连续读取多个扇区，即一次性读取一个”块”（block）。这种由多个扇区组成的”块”，是文件存取的最小单位。”块”的大小，最常见的是4KB，即连续八个 sector组成一个 block。
文件数据都储存在”块”中，那么很显然，我们还必须找到一个地方储存文件的元信息，比如文件的创建者、文件的创建日期、文件的大小等等。这种储存文件元信息的区域就叫做inode，中文译名为”索引节点”。
每一个文件都有对应的inode，里面包含了与该文件有关的一些信息。

inode的数据结构

- 文件的字节数
- 文件拥有者的User ID
- 文件的Group ID
- 文件的读、写、执行权限
- 文件的时间戳，共有三个：ctime指inode上一次变动的时间，mtime指文件内容上一次变动的时间，atime指文件上一次打开的时间。
- 链接数，即有多少文件名指向这个inode

- 索引项，文件数据block的位置

可以用stat命令，查看某个文件的inode信息：

```
$ stat debug.txt
  File: `debug.txt'
  Size: 4444            Blocks: 16         IO Block: 4096   regular file
Device: 801h/2049d      Inode: 387079      Links: 1
Access: (0644/-rw-r--r--)  Uid: ( 1000/  ruanyf)   Gid: ( 1000/  ruanyf)
Access: 2011-12-03 22:54:20.659676310 +0800
Modify: 2011-08-20 00:21:15.429968188 +0800
Change: 2011-08-20 00:21:15.429968188 +0800
$
```

inode的ID

每个inode都有一个唯一的ID，操作系统用inode的ID来识别不同的文件。

这里值得重复一遍的是，Unix/Linux系统内部不使用文件名，而使用inode号码来识别文件。对于系统来说，文件名只是inode ID便于识别的别称或者绰号。

表面上，用户通过文件名，打开文件。实际上，系统内部这个过程分成三步：首先，系统找到这个文件名对应的inode ID；其次，通过inode ID，获取inode信息；最后，根据inode信息，找到文件数据所在的block，读出数据。

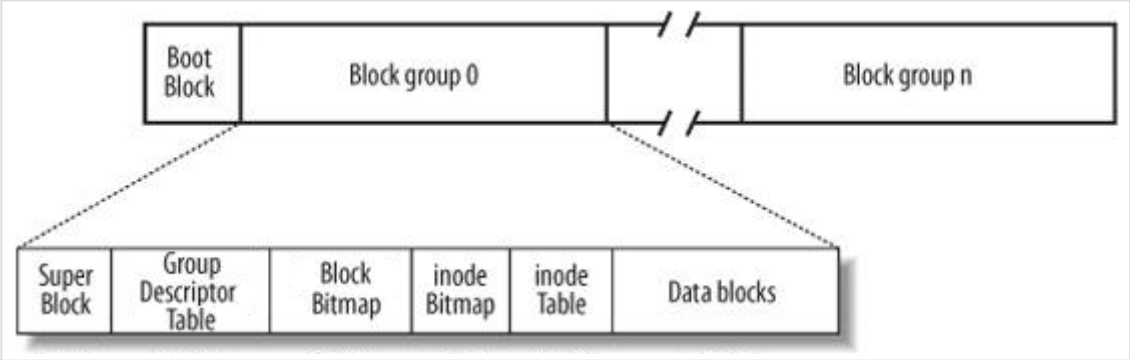
使用ls -i命令，可以看到文件名对应的inode号码：

```
$ ls -i debug.txt
387079 debug.txt
$
```

EXT2文件系统

了解了上面文件目录和文件名路径与inode的对应关系，我们再聊聊操作系统是怎么存储文件数据的。

上面已经提到，为了效率的考虑，操作系统从硬盘上存取数据不会以扇区（512B）为单位，而是一次性存取多个扇区为一个操作的最小单位，在EXT2文件系统中，这个最小单位称为Block（一般为1KB）。不管是文件数据还是目录数据，最终都是存储在Block的数据结构中。另外，n个Bolck块会合成一个块组（Block Group）。ext2文件系统将整个分区划成若干个同样大小的块组（Block Group），每个块组都由以下部分组成。



我们主要需要了解的是inode表和数据块：

inode表

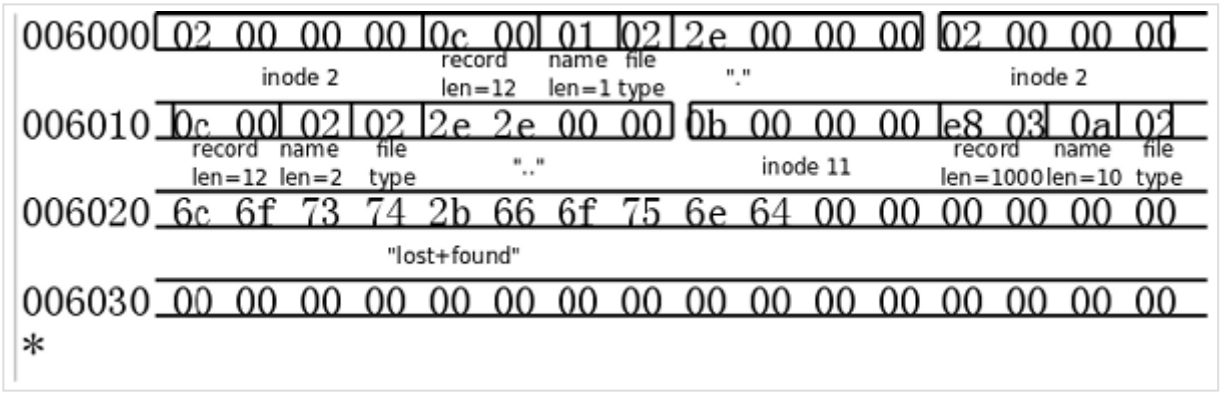
其实一个inode的数据就是存在一个Block中的，但是因为它比较特殊，为了便于操作系统能快速的查找，所以我们把它和普通的Block分开。inode中所存储的数据如上面inode的数据结构中介绍过，这里不再赘述。inode表占多少个块在格式化时就要决定并写入块组描述符中，mke2fs格式化工具的默认策略是一个块组有多少个8KB就分配多少个inode。由于数据块占了整个块组的绝大部分，也可以近似认为数据块有多少个8KB就分配多少个inode，换句话说，如果平均每个文件的大小是8KB，当分区存满的时候inode表会得到比较充分的利用，数据块也不浪费。如果这个分区存的都是很大的文件（比如电影），则数据块用完的时候inode会有一些浪费，如果这个分区存的都是很小的文件（比如源代码），则有可能数据块还没用完inode就已经用完了，数据块可能有很大的浪费。如果用户在格式化时能够对这个分区以后要存储的文件大小做一个预测，也可以用mke2fs的-i参数手动指定每多少个字节分配一个inode。

数据块

根据不同的文件类型有以下几种情况

- 对于常规文件，文件的数据存储在数据块中。
- 对于目录，该目录下的所有**文件名和目录名**存储在数据块中，注意文件名保存在它所在目录的数据块中，除文件名之外，ls -l命令看到的其它信息都保存在该文件的inode中。注意这个概念：目录也是一种文件，是一种特殊类型的文件。
- 对于符号链接，如果目标路径名较短则直接保存在inode中以便更快地查找，如果目标路径名较长则分配一个数据块来保存。
- 设备文件、FIFO和socket等特殊文件没有数据块，设备文件的主设备号和次设备号保存在inode中。

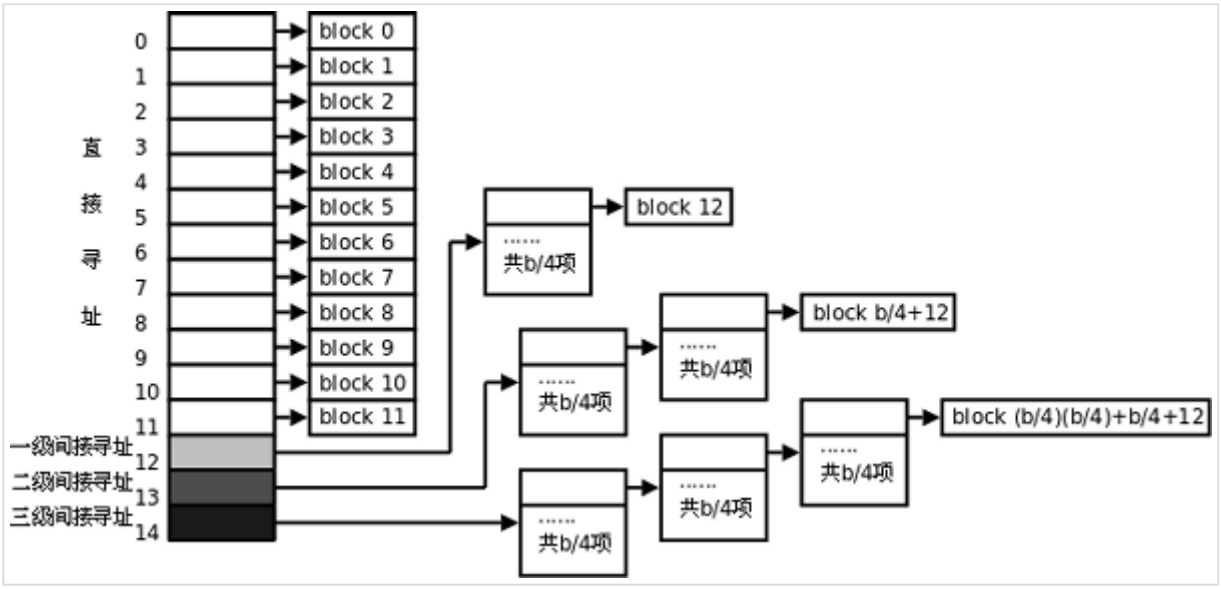
目录也是一种文件，它的数据也是保存在Block中的，只不过它是一种特殊的文件，只存储了当前目录下所有的文件名或目录名以及其对应的inode ID。下图是一个目录文件的存储示例，包括inode ID，记录长度，文件名长度，文件类型以及文件名字符串。一个目录会占用一个或多个Block，所以目录的大小都是block大小的倍数。



文件的数据是存储在几个不一定连续的Block中，那么Block究竟是怎样寻址的呢？这里就要用到inode数据结构中的索引项——Blocks[i]。

事实上，这样的索引项一共有15个，从Blocks[0]到Blocks[14]，每个索引项占4字节。前12个索引项都表示块编号，例如上面的例子中Blocks[0]字段保存着24，就表示第24个块是该文件的数据块，如果块大小是1KB，这样可以表示从0字节到12KB的文件。如果剩下的三个索引项Blocks[12]到Blocks[14]也是这么用的，就只能表示最大15KB的文件了，这是远远不够的，事实上，剩下的三个索引项都是间接索引。

索引项Blocks[12]所指向的块并非数据块，而是称为间接寻址块（Indirect Block），其中存放的都是类似Blocks[0]这种索引项，再由索引项指向数据块。设块大小是b，那么一个间接寻址块中可以存放b/4个索引项，指向b/4个数据块。所以如果把Blocks[0]到Blocks[12]都用上，最多可以表示b/4+12个数据块，对于块大小是1K的情况，最大可表示268K的文件。如下图所示，注意文件的数据块编号是从0开始的，Blocks[0]指向第0个数据块，Blocks[11]指向第11个数据块，Blocks[12]所指向的间接寻址块的第一个索引项指向第12个数据块，依此类推。



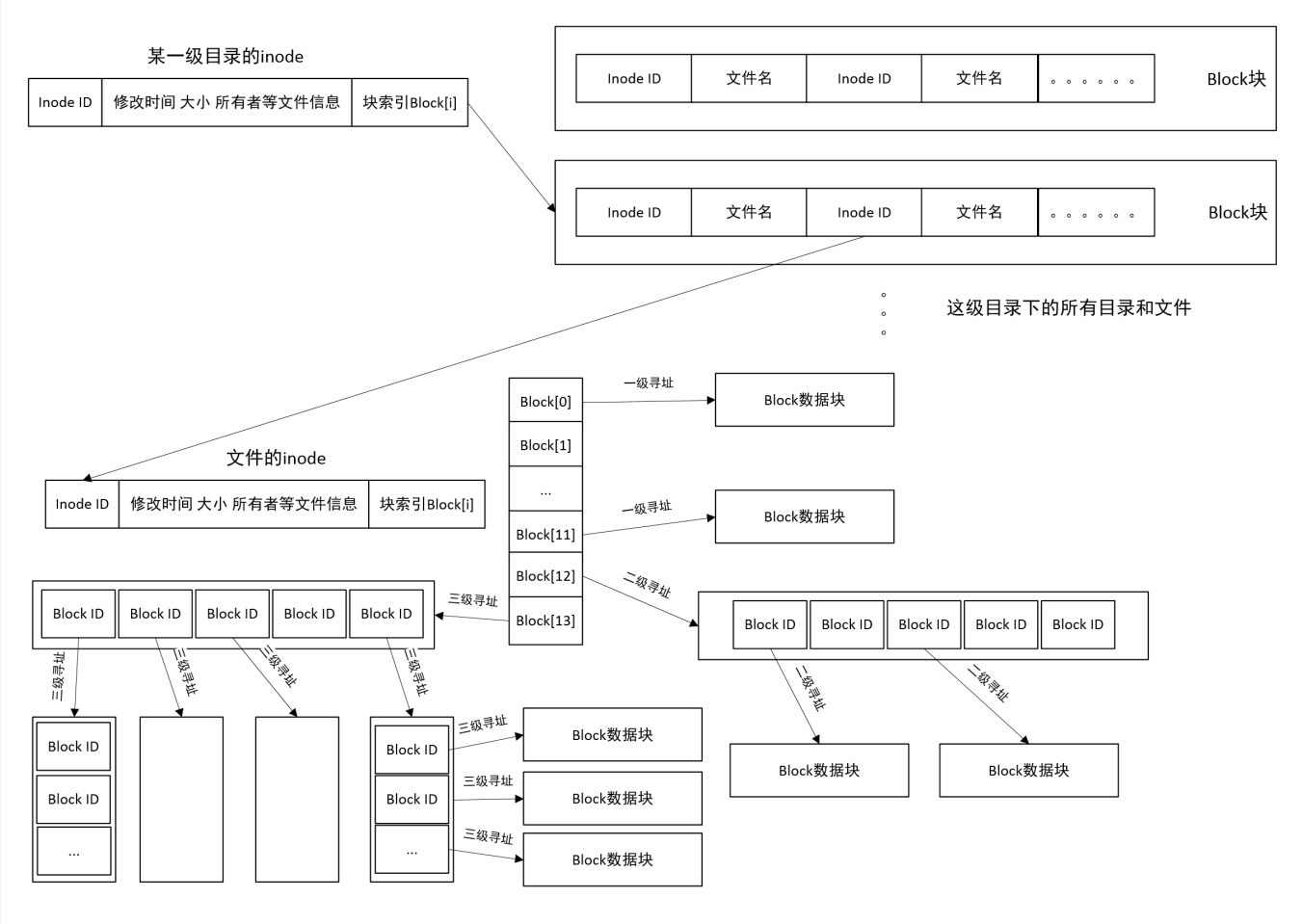
从上图可以看出，索引项Blocks[13]指向两级的间接寻址块，最多可表示(b/4)2+b/4+12个数据块，对于1K的块大小最大可表示64.26MB的文件。索引项Blocks[14]指向三级的间接寻址块，最多可表示(b/4)3+(b/4)2+b/4+12个数据块，对于1K的块大小最大可表示16.06GB的文件。

可见，这种寻址方式对于访问不超过12个数据块的小文件是非常快的，访问文件中的任意数据只需要两次读盘操作，一次读inode（也就是读索引项）一次读数据块。而访问大文件中的数据则需要最多五次读盘操作：

inode、一级间接寻址块、二级间接寻址块、三级间接寻址块、数据块。实际上，磁盘中的inode和数据块往往已经被内核缓存了，读大文件的效率也不会太低。

我们写出一个读文件的完整流程，我们以/root/my/path/test.txt文件的读取为例。

1. 从跟目录“/”开始遍历迭代，找到根目录的inode。
2. 读取根目录inode的block块，遍历所有的文件名，找到下一级目录my及其目录inode。
3. 重复1和2直到找到test.txt的inode。
4. 读出test.txt的inode的Block块数据
5. 根据读取文件的偏移量和Block size，从inode的索引项Blocks[i]中，通过一级或多级寻址，找到偏移量对应的Block
6. 读取Block



至此，Linux中ext2文件系统的大致操作就介绍完了。目前，ext文件系统已经推出了第4版。ext4对针对新的SSD设备做了大量的优化。ext3是ext2的一个升级，增加了包括文件系统日志等一些有用的feature。另外，业界还有很多其它的文件系统，如windows的ntfs，sun的zfs，apple的hfs，以及最近IOS升级后大热的apfs，篇幅有限就不过多的介绍了。

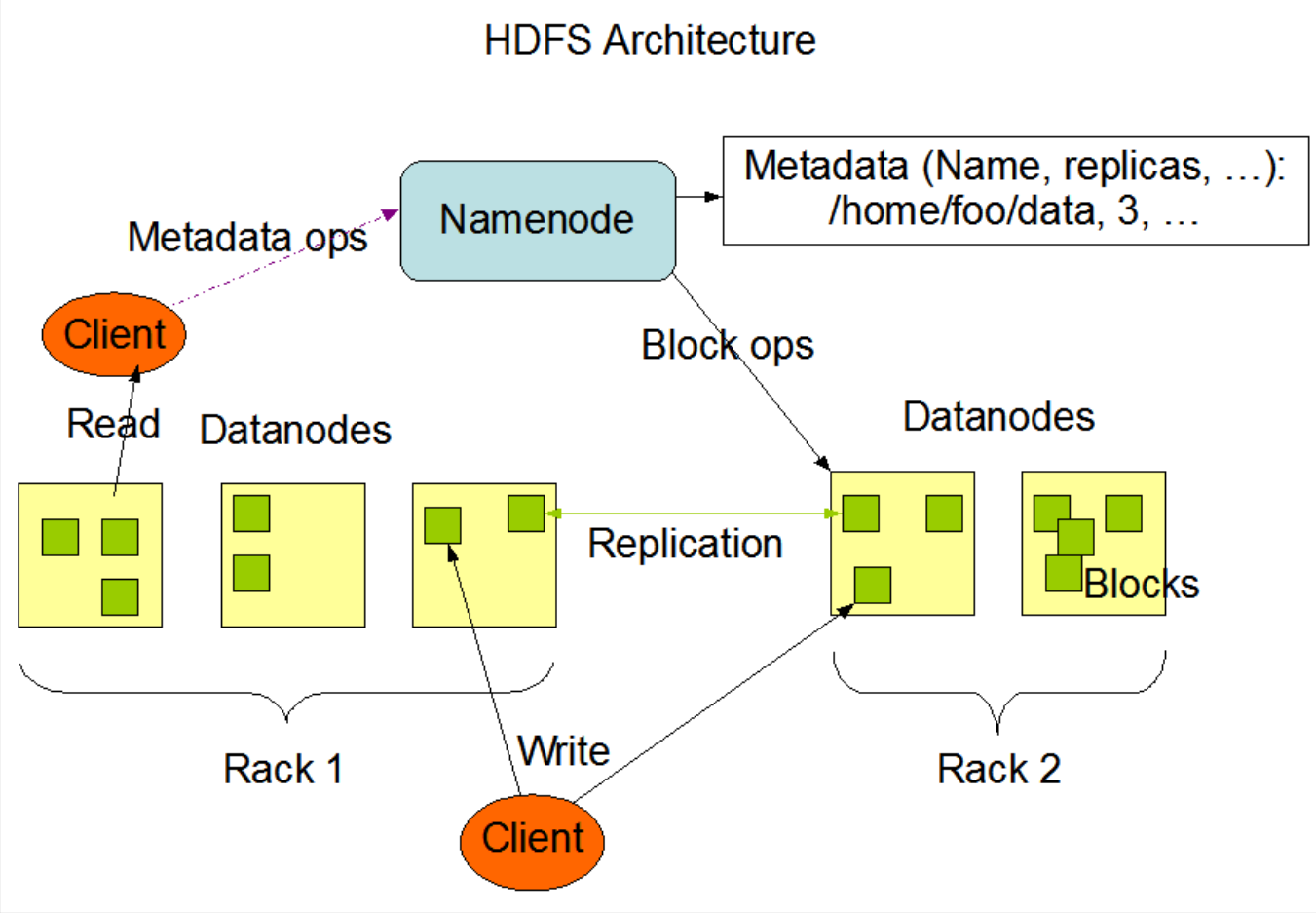
下面我们来看看Hadoop上的分布式文件系统HDFS

HDFS

分布式文件系统HDFS是构建在Linux的文件系统之上的，其虚拟的数据存储介质为Linux文件系统上的文件。

NameNode和DataNode

HDFS采用master/slave架构。一个HDFS集群是由一个Namenode和一定数目的Datanodes组成。Namenode是一个中心服务器，负责管理文件系统的名字空间(namespace)以及客户端对文件的访问。集群中的Datanode一般是一个节点一个，负责管理它所在节点上的存储。HDFS暴露了文件系统的名字空间，用户能够以文件的形式在上面存储数据。从内部看，一个文件其实被分成一个或多个数据块，这些块存储在一组Datanode上。Namenode执行文件系统的名字空间操作，比如打开、关闭、重命名文件或目录。它也负责确定数据块到具体Datanode节点的映射。Datanode负责处理文件系统客户端的读写请求。在Namenode的统一调度下进行数据块的创建、删除和复制。



分布式文件系统的inode

分布式文件系统hdfs借鉴了Linux系统上的设计，文件名和目录与存储实体文件数据的映射也采用了inode的设计。

hdfs的inode类型分为文件型inode和目录型inode，INode为基础抽象类，保存了文件和目录都可能会用到的公共属性。

```
/**
 * We keep an in-memory representation of the file/block hierarchy.
 * This is a base INode class containing common fields for file and
 * directory inodes.
 */
abstract class INode implements Comparable<byte[]> {
  //文件/目录名称
  protected byte[] name;
  //父目录
  protected INodeDirectory parent;
  //最近一次的修改时间
  protected long modificationTime;
  //最近访问时间
  protected long accessTime;
```

INodeFile

INodeFile为文件的inode实现，与Linux文件系统类似，hdfs的文件数据也是以Block为最小的存储单位，而因为hdfs被设计用来存储海量数据，处理的文件都比较大，为了最小化寻址的开销，自然其Block块会设计的相对大一些，默认设置是128M。

block列表中保存了这个文件所有的block，用于文件数据的寻址。在hdfs中，因为InodeFile类型的存储大小不受限制（List为容器），不同于Linux文件系统的block的block size 的设置，所以其block索引的寻址没有设计的如Linux文件系统那么复杂（一级寻址和多级寻址）。

另外，基于分布式文件系统的设计思想，所有文件都会被切分成若干数据块分布在数据节点（DataNode）上存储，同时每个数据块会冗余备份到不同的数据节点上（机架感知，默认保存3份）。

```
class INodeFile extends INode {
  static final FsPermission UMASK = FsPermission.createImmutable((short)0111);

  //Number of bits for Block size
  //48位存储block数据块的大小
  static final short BLOCKBITS = 48;

  //Header mask 64-bit representation
  //Format: [16 bits for replication][48 bits for PreferredBlockSize]
  //前16位保存副本系数,后48位保存优先块大小,下面的headermask做计算时用
  static final long HEADERMASK = 0xffffL << BLOCKBITS;

  protected long header;
  //文件数据block块
  protected BlockInfo blocks[] = null;
```

INodeDirectory

InodeDirectory为目录的inode实现，其中的children属性为一个List，用来保存目录的中的子目录或子文件节点。这点与Linux的文件系统设计略有差别，目录和文件没有统一用相同的

```
/**
 * Directory INode class.
 */
class INodeDirectory extends INode {
  protected static final int DEFAULT_FILES_PER_DIRECTORY = 5;
  final static String ROOT_NAME = "";

  //保存子目录或子文件
  private List<INode> children;
```

BlockPool

每个DataNode都会维护一个BlockPool，用以提供Block的管理能力。DataNode通过BlockPool给NameNode提供了Block管理的功能，但是NameNode从不主动的去请求DataNode去做何种动作，而是DataNode针对每个BlockPool都维护一个与自己归属的NameNode之间心跳线程，定期的向NameNode汇报自身的状态，在NameNode返回的心跳值中会携带相关的Command命令信息，从而完成NameNode对DataNode控制。

寻址流程

hdfs的文件存取流程相对比较简单，首先在NameNode节点上，通过文件路径从根目录一级一级遍历目录的inode直至找到文件的inode，再通过文件inode的block索引List，经过偏移量的计算找到对应的block块。最后，通过在NameNode上查询BlockPool和Block的注册信息，最终找到对应的Block块。
看似整个寻址的流程比Linux文件系统简化了不少，这是源于HDFS建立在Linux的文件系统之上，提供大文件的分布式文件管理。其架构的设计更多的着重于数据分布式问题的解决。比如分布式数据的一致性，可用性，分区容忍度（CAP三原则）。
举几个简单的方便。比如，如何在写入数据的时候同时写入三份数据备份（用到pipe line管道的建立）？如何分配三份数据在服务器、机架甚至机房的位置（硬盘、服务器、机架以及机房感知）？如何DataNode的注册心跳、应对DataNode节点的扩容、缩减以及自动的故障转移，都是作为一个分布式文件系统更需要关注的方面。

结语

可以看出，HDFS的设计在数据块、索引查找等很多方面借鉴了Linux文件系统的设计思想。在技术发展呈现爆炸式增长的今天，没有任何一个软件系统是从基础的造轮子开始的，都是需要在现有的技术栈上，通过使用现有的成熟组件亦或是借鉴前辈们在各类基础系统中的设计思想，来设计和最终实现新的软件系统。所以，我们经常去深入研究一些基础系统的原理、实现方法和优劣异同，对今后的架构和开发工作一定会大有裨益的。